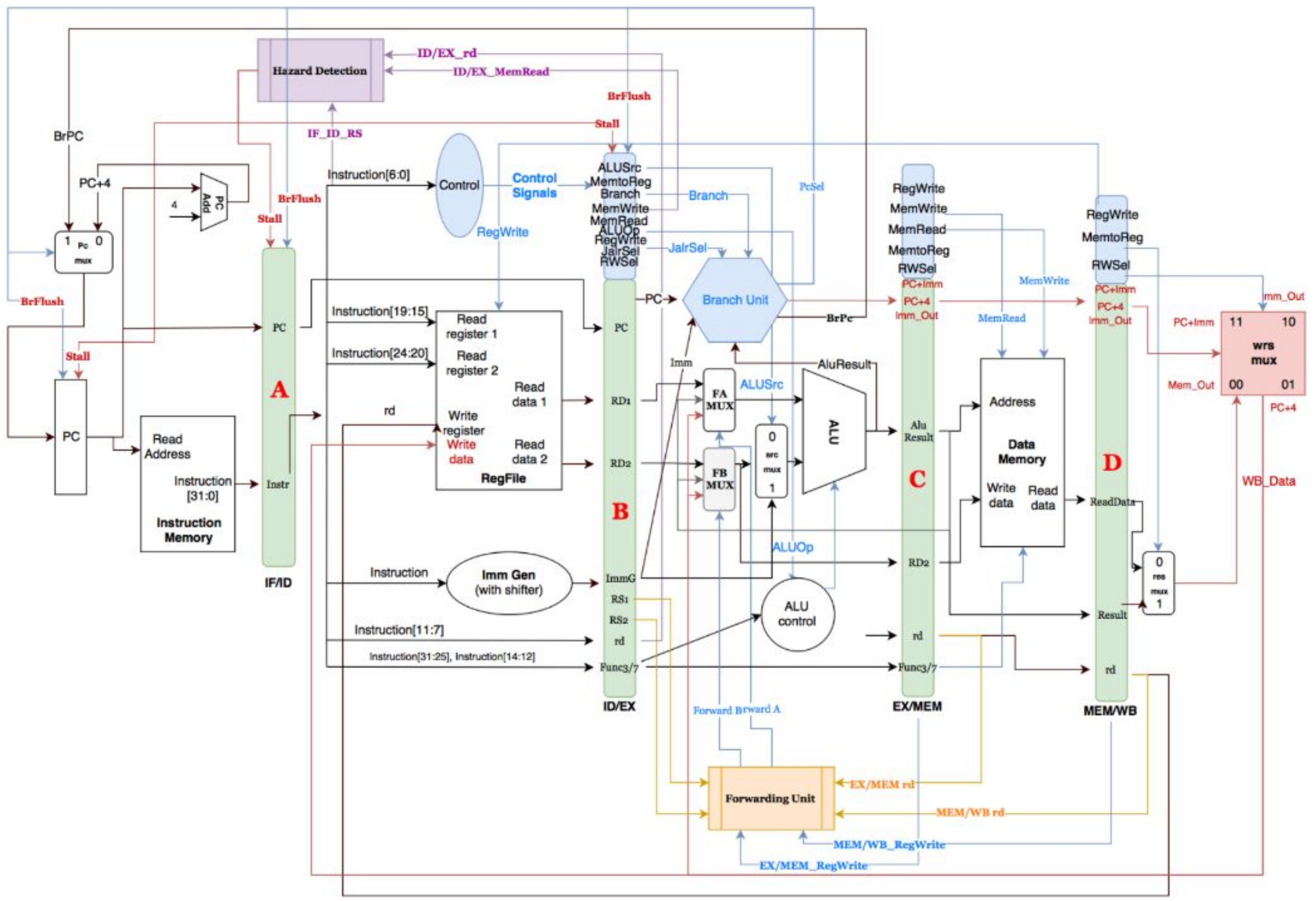
# 6863 Formal Verification Project - a RISC-V design

Group Member: Ye Hang, Junfeng Zou, Yunyang Lu

## Motivation

The RISC(Reduced Instruction Set Computing)-V processor is an open-source, modular instruction set architecture which is widely researched and implemented in various applications. Despite its popularity, the formal verification of RISC-V designs are not always available. This project focuses on verifying key functionalities of a RISC-V processor pipeline architecture implemented in SystemVerilog, which is available in https://github.com/estufa-cin-ufpe/RISC-V-Pipeline.

# Overview



Hazard Detection
ID/EX_rd
ID/EX_MemRead
BrFlush
Stall
IF_ID_RS
BrPC
PC+4
4
PC Add
1 Pc mux 0
BrFlush
Stall
PC
Read Address
Instruction [31:0]
Instruction Memory
A
PC
Instr
IF/ID
Instruction[6:0]
Control
Control Signals
RegWrite
Instruction[19:15]
Instruction[24:20]
rd
Read register 1
Read register 2
Read data 1
Write register
Write data
Read data 2
RegFile
Instruction
Imm Gen (with shifter)
Instruction[11:7]
Instruction[31:25], Instruction[14:12]
ALUSrc
MemtoReg
Branch
MemWrite
MemRead
ALUOp
RegWrite
JalrSel
RWSel
PC
RD1
RD2
B
ImmG
RS1
RS2
rd
Func3/7
ID/EX
Branch
JalrSel
PcSel
Branch Unit
BrPc
Imm
AluResult
FA MUX
FB MUX
ALUSrc
0 src mux 1
ALU
ALUOp
ALU control
Forward A
Forward B
Forwarding Unit
EX/MEM rd
MEM/WB rd
MEM/WB_RegWrite
EX/MEM_RegWrite
RegWrite
MemWrite
MemRead
MemtoReg
RWSel
PC+Imm
PC+4
Imm_Out
Alu Result
C
RD2
rd
Func3/7
EX/MEM
MemWrite
MemRead
Address
Data Memory
Write data
Read data
RegWrite
MemtoReg
RWSel
PC+Imm
PC+4
Imm_Out
D
ReadData
Result
rd
MEM/WB
0 res mux 1
Imm_Out
PC+Imm
11 10
wrs mux
Mem_Out
00 01
PC+4
WB_Data

## Methodology

Overall - bottom up approach: First analyze the components, then analyze the system-wise functionality.

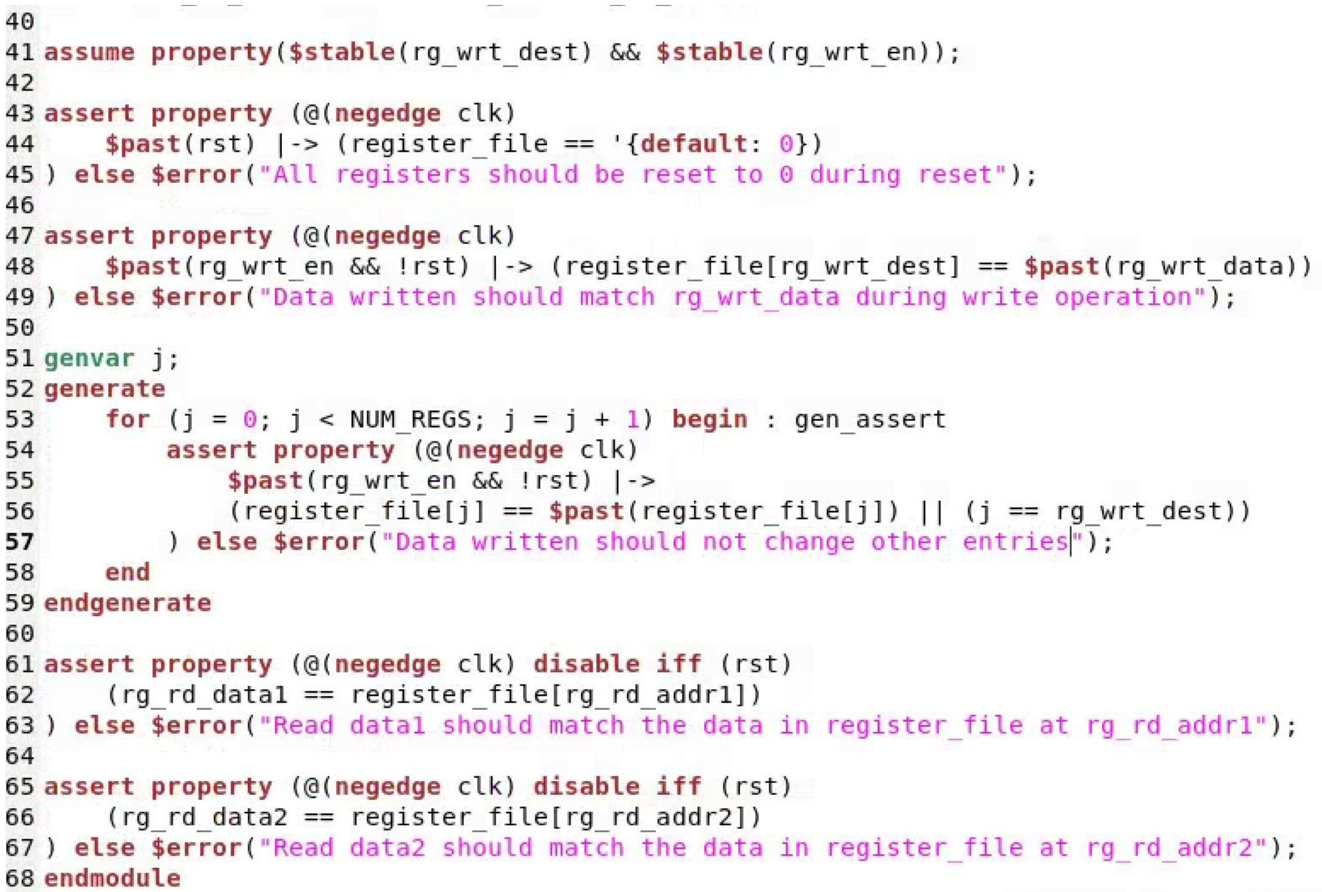
For each component,

- Write properties
- Convert properties into assertions, assumptions, etc. in .sv file
- Run the formal verification tool (Jasper Gold)
- Check the result and prove/disprove the component functionality

## Register file

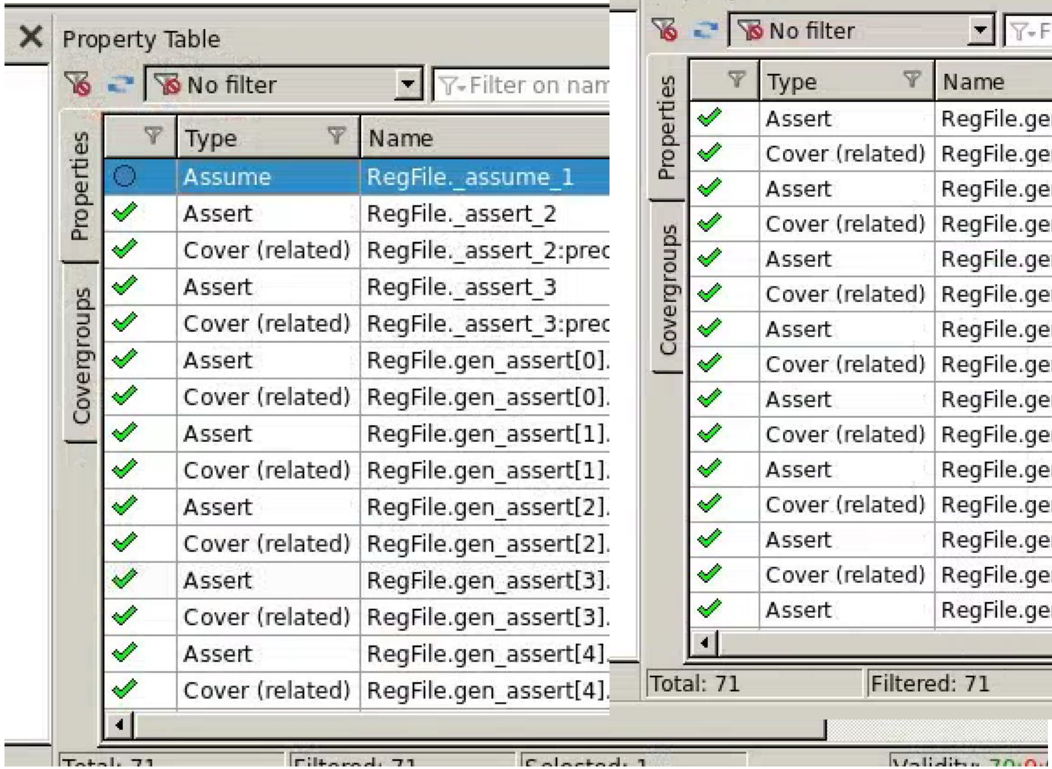
1. Safety Property
    a. When there is a write request, the registers in other addresses should not change their value.
    b. The values stored in the register file are cleared to 0 after the reset signal is received.
2. Liveness Property
    a. If there is a write request, the register in the designated address will eventually update to the write data value (or be updated to the correct value in expected cycles).
    b. A change in read register address(es) should switch the correct corresponding Read data output in one clock cycle.

## Register file



```

assume property($stable(rg_wrt_dest) && $stable(rg_wrt_en));

assert property (@(negedge clk)
    $past(rst) |-> (register_file == '{default: 0})
) else $error("All registers should be reset to 0 during reset");

assert property (@(negedge clk)
    $past(rg_wrt_en && !rst) |-> (register_file[rg_wrt_dest] == $past(rg_wrt_data))
) else $error("Data written should match rg_wrt_data during write operation");

genvar j;
generate
    for (j = 0; j < NUM_REGS; j = j + 1) begin : gen_assert
        assert property (@(negedge clk)
            $past(rg_wrt_en && !rst) |->
            (register_file[j] == $past(register_file[j]) || (j == rg_wrt_dest))
        ) else $error("Data written should not change other entries");
    end
endgenerate

assert property (@(negedge clk) disable iff (rst)
    (rg_rd_data1 == register_file[rg_rd_addr1])
) else $error("Read data1 should match the data in register_file at rg_rd_addr1");

assert property (@(negedge clk) disable iff (rst)
    (rg_rd_data2 == register_file[rg_rd_addr2])
) else $error("Read data2 should match the data in register_file at rg_rd_addr2");
endmodule
```

# Register file

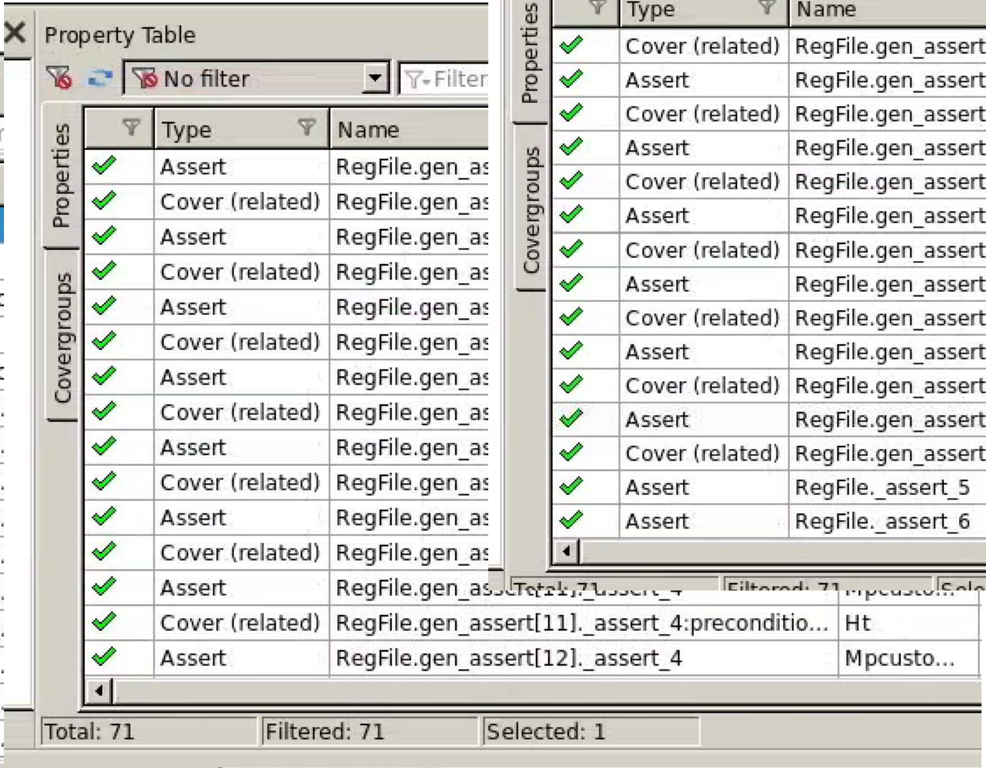


Property Table

No filter | Filter on nam

| | Type | Name |
|---|---|---|
| ○ | Assume | RegFile._assume_1 |
| ✓ | Assert | RegFile._assert_2 |
| ✓ | Cover (related) | RegFile._assert_2:prec |
| ✓ | Assert | RegFile._assert_3 |
| ✓ | Cover (related) | RegFile._assert_3:prec |
| ✓ | Assert | RegFile.gen_assert[0]. |
| ✓ | Cover (related) | RegFile.gen_assert[0]. |
| ✓ | Assert | RegFile.gen_assert[1]. |
| ✓ | Cover (related) | RegFile.gen_assert[1]. |
| ✓ | Assert | RegFile.gen_assert[2]. |
| ✓ | Cover (related) | RegFile.gen_assert[2]. |
| ✓ | Assert | RegFile.gen_assert[3]. |
| ✓ | Cover (related) | RegFile.gen_assert[3]. |
| ✓ | Assert | RegFile.gen_assert[4]. |
| ✓ | Cover (related) | RegFile.gen_assert[4]. |

Total: 71 | Filtered: 71 | Selected: 1 | Validity: 70:0

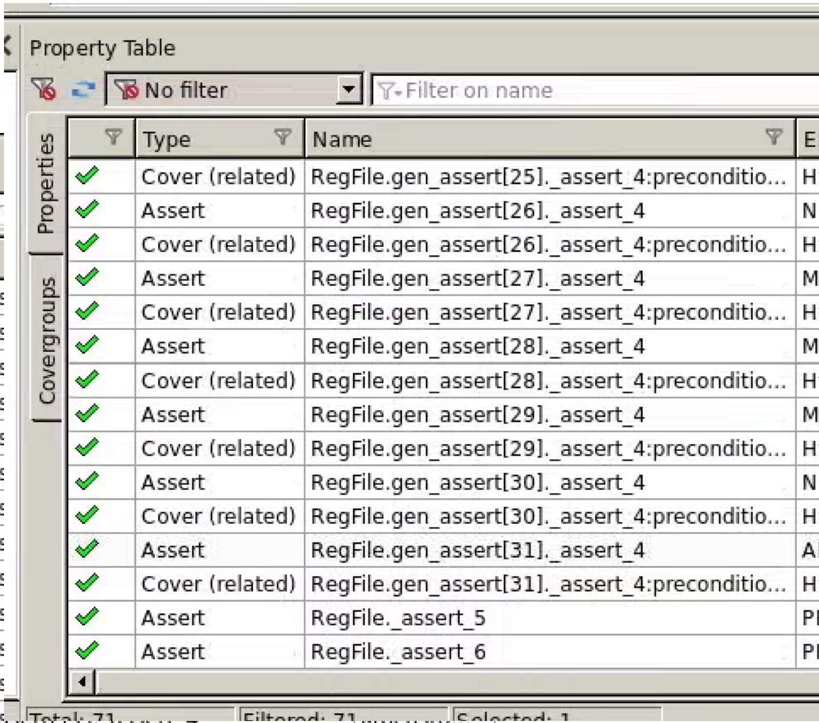


Property Table

No filter | Filter

| | Type | Name | |
|---|---|---|---|
| ✓ | Assert | RegFile.gen_as | |
| ✓ | Cover (related) | RegFile.gen_as | |
| ✓ | Assert | RegFile.gen_as | |
| ✓ | Cover (related) | RegFile.gen_as | |
| ✓ | Assert | RegFile.gen_as | |
| ✓ | Cover (related) | RegFile.gen_as | |
| ✓ | Assert | RegFile.gen_as | |
| ✓ | Cover (related) | RegFile.gen_as | |
| ✓ | Assert | RegFile.gen_as | |
| ✓ | Cover (related) | RegFile.gen_as | |
| ✓ | Assert | RegFile.gen_as | |
| ✓ | Cover (related) | RegFile.gen_as | |
| ✓ | Assert | RegFile.gen_as | |
| ✓ | Cover (related) | RegFile.gen_assert[11]._assert_4:preconditio... | Ht |
| ✓ | Assert | RegFile.gen_assert[12]._assert_4 | Mpcusto... |

Total: 71 | Filtered: 71 | Selected: 1



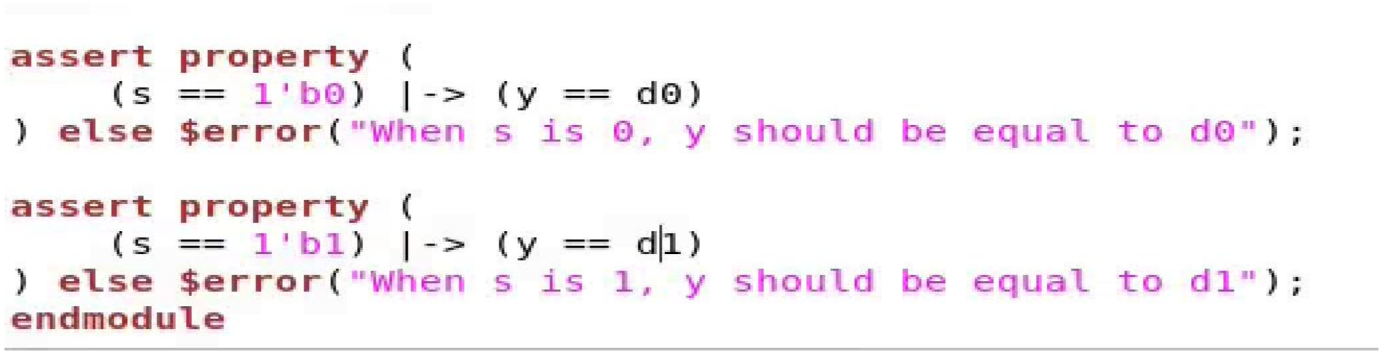
Property Table

No filter | Filter on name

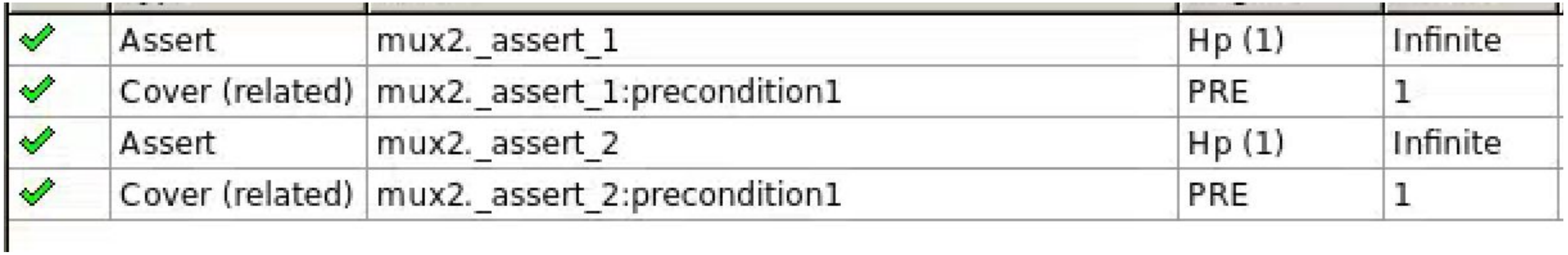
| | Type | Name | E |
|---|---|---|---|
| ✓ | Cover (related) | RegFile.gen_assert[25]._assert_4:preconditio... | H |
| ✓ | Assert | RegFile.gen_assert[26]._assert_4 | N |
| ✓ | Cover (related) | RegFile.gen_assert[26]._assert_4:preconditio... | H |
| ✓ | Assert | RegFile.gen_assert[27]._assert_4 | M |
| ✓ | Cover (related) | RegFile.gen_assert[27]._assert_4:preconditio... | H |
| ✓ | Assert | RegFile.gen_assert[28]._assert_4 | M |
| ✓ | Cover (related) | RegFile.gen_assert[28]._assert_4:preconditio... | H |
| ✓ | Assert | RegFile.gen_assert[29]._assert_4 | M |
| ✓ | Cover (related) | RegFile.gen_assert[29]._assert_4:preconditio... | H |
| ✓ | Assert | RegFile.gen_assert[30]._assert_4 | N |
| ✓ | Cover (related) | RegFile.gen_assert[30]._assert_4:preconditio... | H |
| ✓ | Assert | RegFile.gen_assert[31]._assert_4 | A |
| ✓ | Cover (related) | RegFile.gen_assert[31]._assert_4:preconditio... | H |
| ✓ | Assert | RegFile._assert_5 | P |
| ✓ | Assert | RegFile._assert_6 | P |

Total: 71 | Filtered: 71 | Selected: 1

## MUX



```
assert property (
    (s == 1'b0) |-> (y == d0)
) else $error("When s is 0, y should be equal to d0");

assert property (
    (s == 1'b1) |-> (y == d1)
) else $error("When s is 1, y should be equal to d1");
endmodule
```

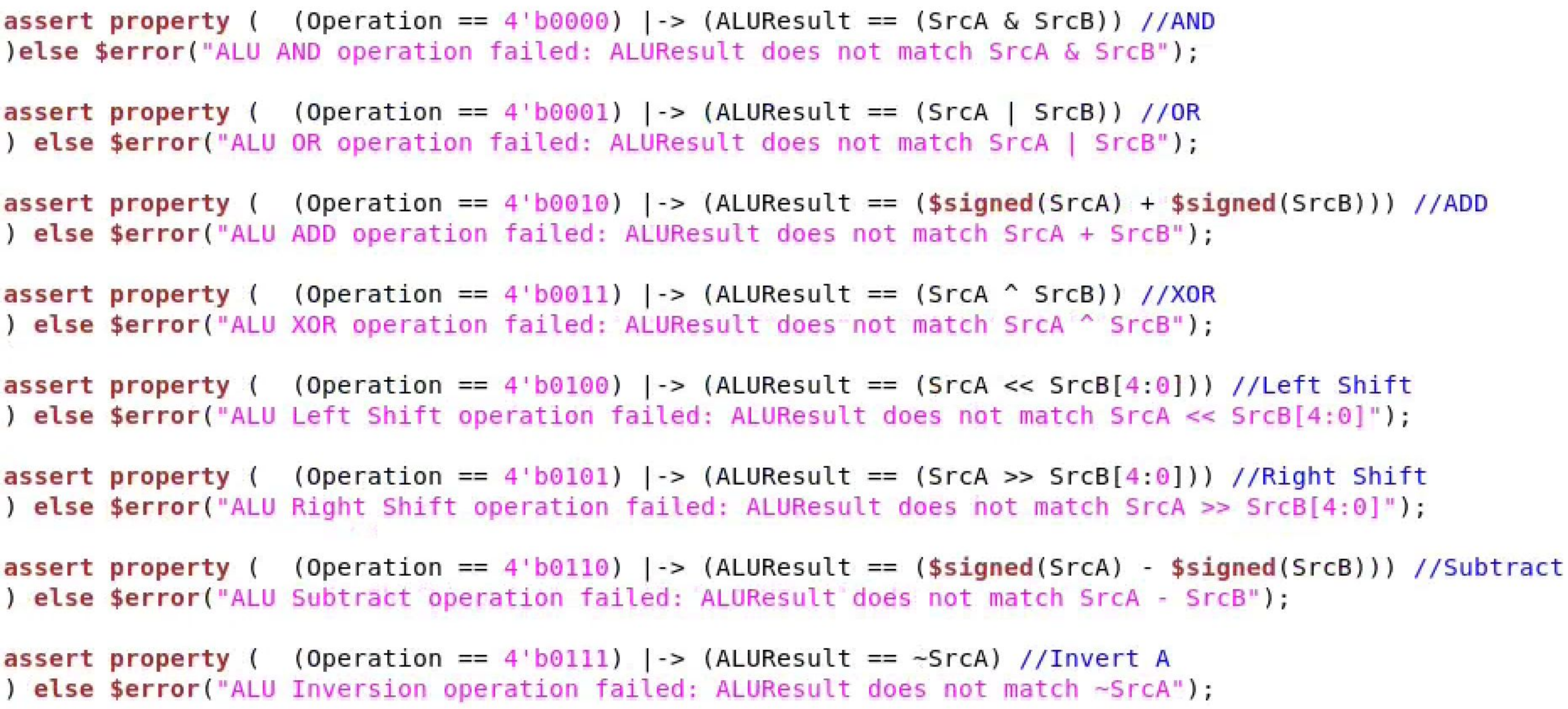


| | | | | |
|---|---|---|---|---|
| ✓ | Assert | mux2._assert_1 | Hp (1) | Infinite |
| ✓ | Cover (related) | mux2._assert_1:precondition1 | PRE | 1 |
| ✓ | Assert | mux2._assert_2 | Hp (1) | Infinite |
| ✓ | Cover (related) | mux2._assert_2:precondition1 | PRE | 1 |

## ALU

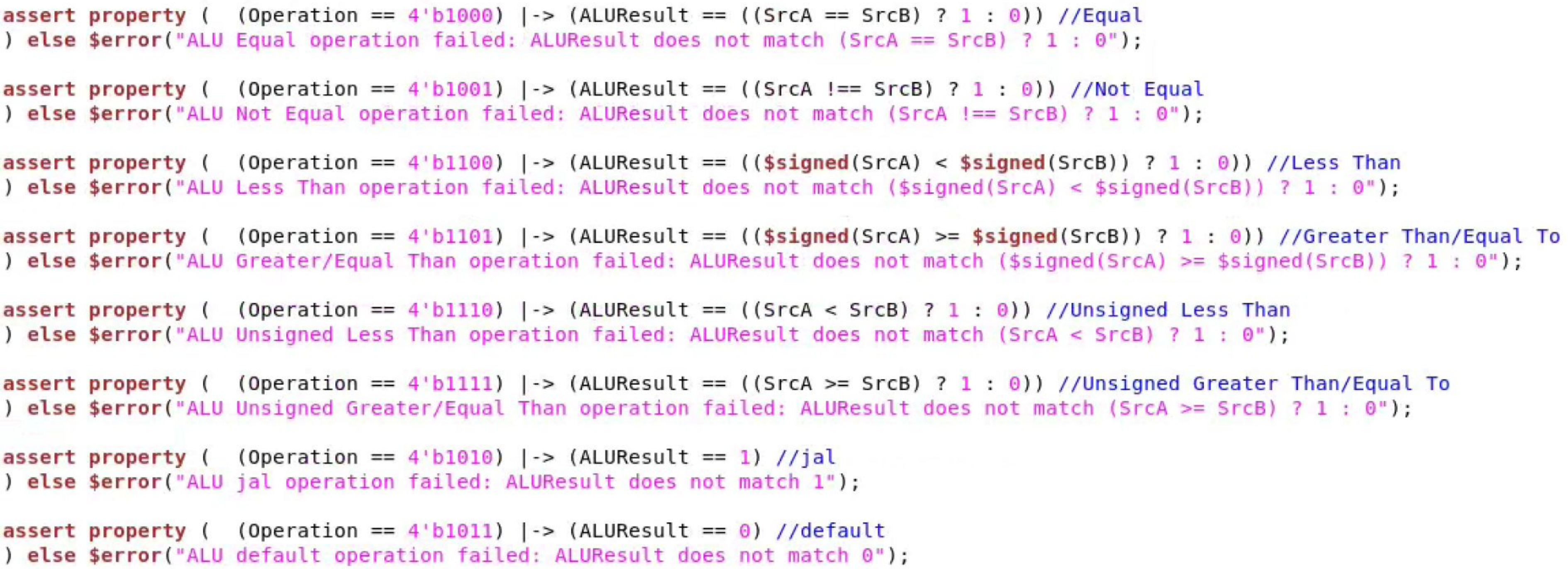
Safety property:
Each function of the ALU works correctly (and, or, add, xor, shift left, shift right, subtract, invert, equal, not equal, less than, greater/equal than, unsigned less than, unsigned greater/equal than, jal, default).

# ALU



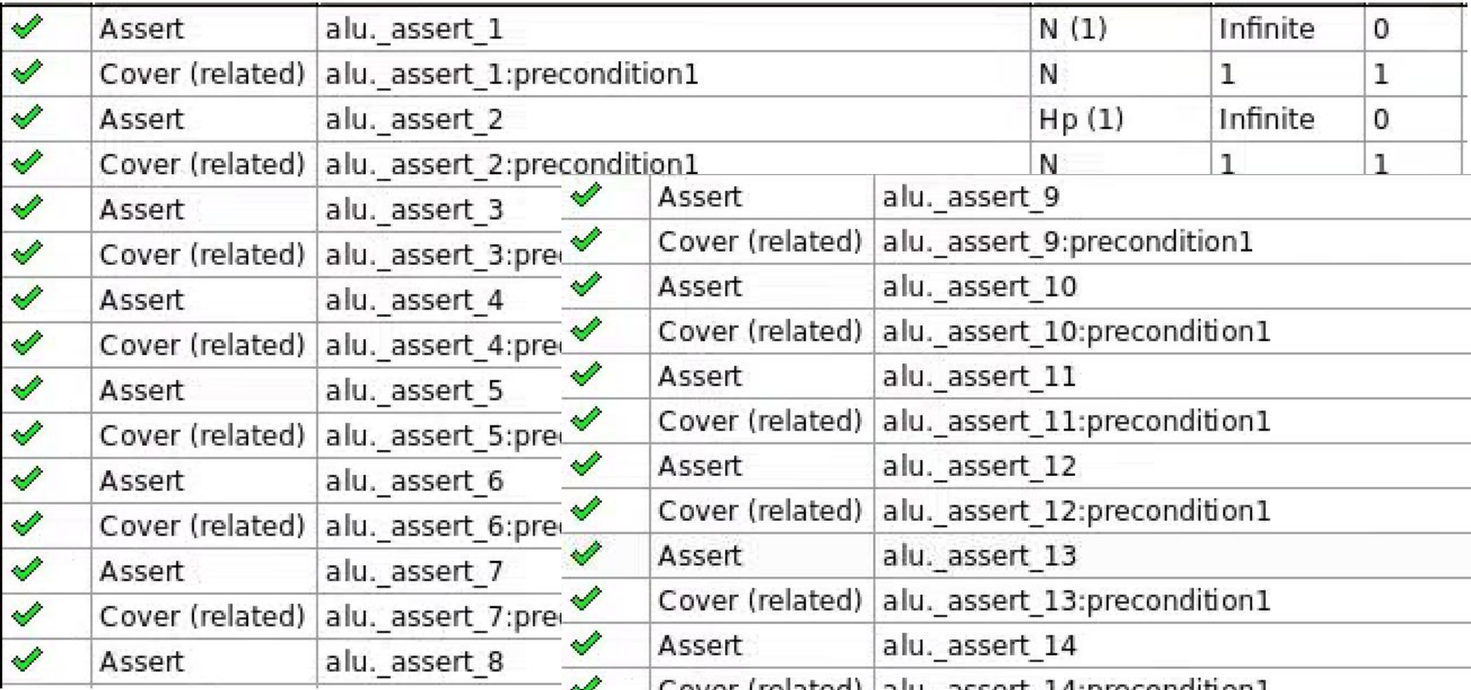
```
assert property (  (Operation == 4'b0000) |-> (ALUResult == (SrcA & SrcB)) //AND
)else $error("ALU AND operation failed: ALUResult does not match SrcA & SrcB");

assert property (  (Operation == 4'b0001) |-> (ALUResult == (SrcA | SrcB)) //OR
) else $error("ALU OR operation failed: ALUResult does not match SrcA | SrcB");

assert property (  (Operation == 4'b0010) |-> (ALUResult == ($signed(SrcA) + $signed(SrcB))) //ADD
) else $error("ALU ADD operation failed: ALUResult does not match SrcA + SrcB");

assert property (  (Operation == 4'b0011) |-> (ALUResult == (SrcA ^ SrcB)) //XOR
) else $error("ALU XOR operation failed: ALUResult does not match SrcA ^ SrcB");

assert property (  (Operation == 4'b0100) |-> (ALUResult == (SrcA << SrcB[4:0])) //Left Shift
) else $error("ALU Left Shift operation failed: ALUResult does not match SrcA << SrcB[4:0]");

assert property (  (Operation == 4'b0101) |-> (ALUResult == (SrcA >> SrcB[4:0])) //Right Shift
) else $error("ALU Right Shift operation failed: ALUResult does not match SrcA >> SrcB[4:0]");

assert property (  (Operation == 4'b0110) |-> (ALUResult == ($signed(SrcA) - $signed(SrcB))) //Subtract
) else $error("ALU Subtract operation failed: ALUResult does not match SrcA - SrcB");

assert property (  (Operation == 4'b0111) |-> (ALUResult == ~SrcA) //Invert A
) else $error("ALU Inversion operation failed: ALUResult does not match ~SrcA");
```

## ALU

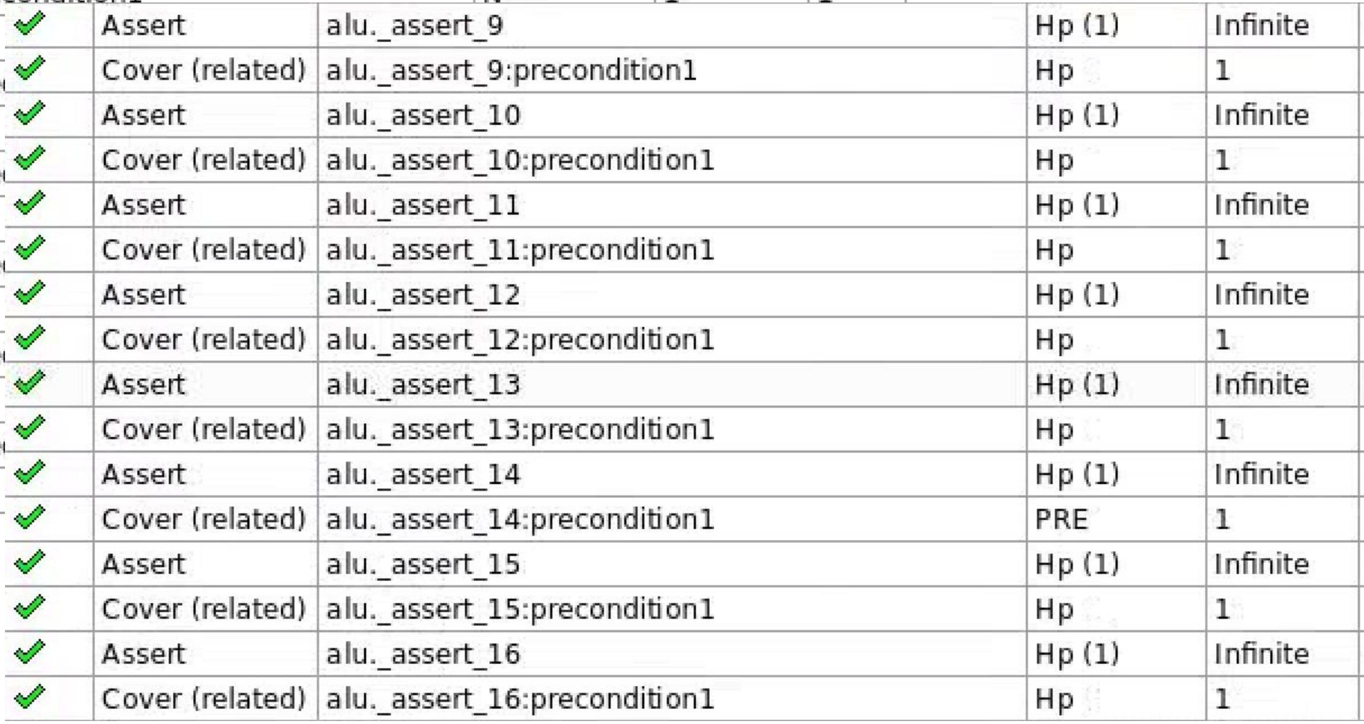


```
assert property (  (Operation == 4'b1000) |-> (ALUResult == ((SrcA == SrcB) ? 1 : 0)) //Equal
) else $error("ALU Equal operation failed: ALUResult does not match (SrcA == SrcB) ? 1 : 0");

assert property (  (Operation == 4'b1001) |-> (ALUResult == ((SrcA !== SrcB) ? 1 : 0)) //Not Equal
) else $error("ALU Not Equal operation failed: ALUResult does not match (SrcA !== SrcB) ? 1 : 0");

assert property (  (Operation == 4'b1100) |-> (ALUResult == (($signed(SrcA) < $signed(SrcB)) ? 1 : 0)) //Less Than
) else $error("ALU Less Than operation failed: ALUResult does not match ($signed(SrcA) < $signed(SrcB)) ? 1 : 0");

assert property (  (Operation == 4'b1101) |-> (ALUResult == (($signed(SrcA) >= $signed(SrcB)) ? 1 : 0)) //Greater Than/Equal To
) else $error("ALU Greater/Equal Than operation failed: ALUResult does not match ($signed(SrcA) >= $signed(SrcB)) ? 1 : 0");

assert property (  (Operation == 4'b1110) |-> (ALUResult == ((SrcA < SrcB) ? 1 : 0)) //Unsigned Less Than
) else $error("ALU Unsigned Less Than operation failed: ALUResult does not match (SrcA < SrcB) ? 1 : 0");

assert property (  (Operation == 4'b1111) |-> (ALUResult == ((SrcA >= SrcB) ? 1 : 0)) //Unsigned Greater Than/Equal To
) else $error("ALU Unsigned Greater/Equal Than operation failed: ALUResult does not match (SrcA >= SrcB) ? 1 : 0");

assert property (  (Operation == 4'b1010) |-> (ALUResult == 1) //jal
) else $error("ALU jal operation failed: ALUResult does not match 1");

assert property (  (Operation == 4'b1011) |-> (ALUResult == 0) //default
) else $error("ALU default operation failed: ALUResult does not match 0");
```

# ALU



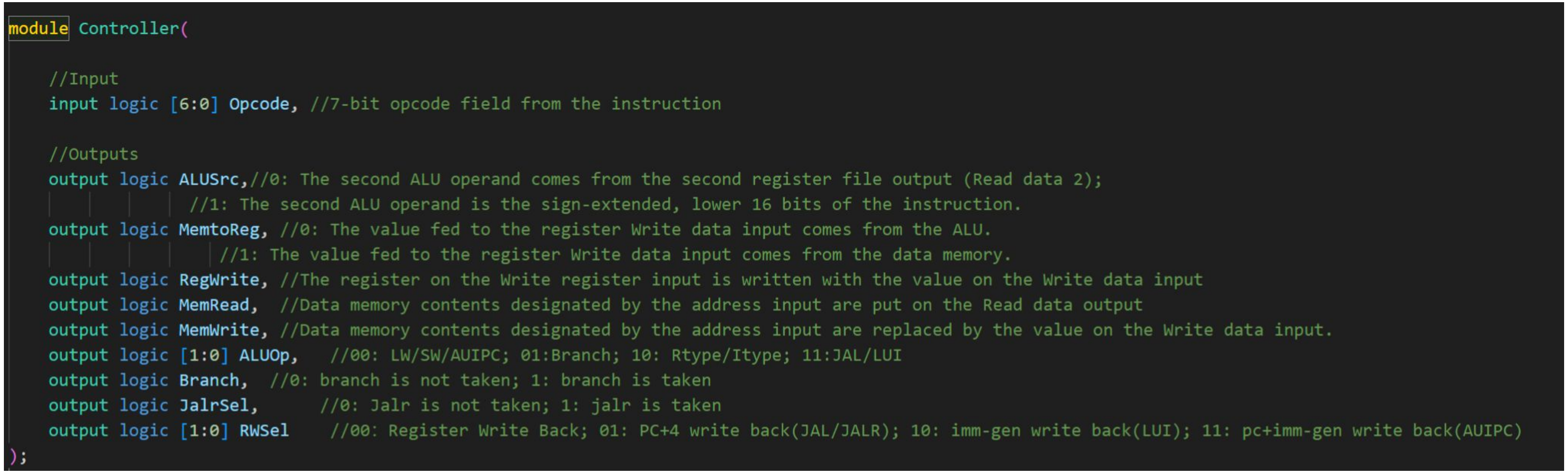
| | | | | | |
|---|---|---|---|---|---|
| ✓ | Assert | alu._assert_1 | N (1) | Infinite | 0 |
| ✓ | Cover (related) | alu._assert_1:precondition1 | N | 1 | 1 |
| ✓ | Assert | alu._assert_2 | Hp (1) | Infinite | 0 |
| ✓ | Cover (related) | alu._assert_2:precondition1 | N | 1 | 1 |
| ✓ | Assert | alu._assert_3 | | | |
| ✓ | Cover (related) | alu._assert_3:pre | | | |
| ✓ | Assert | alu._assert_4 | | | |
| ✓ | Cover (related) | alu._assert_4:pre | | | |
| ✓ | Assert | alu._assert_5 | | | |
| ✓ | Cover (related) | alu._assert_5:pre | | | |
| ✓ | Assert | alu._assert_6 | | | |
| ✓ | Cover (related) | alu._assert_6:pre | | | |
| ✓ | Assert | alu._assert_7 | | | |
| ✓ | Cover (related) | alu._assert_7:pre | | | |
| ✓ | Assert | alu._assert_8 | | | |



| | | | | |
|---|---|---|---|---|
| ✓ | Assert | alu._assert_9 | Hp (1) | Infinite |
| ✓ | Cover (related) | alu._assert_9:precondition1 | Hp | 1 |
| ✓ | Assert | alu._assert_10 | Hp (1) | Infinite |
| ✓ | Cover (related) | alu._assert_10:precondition1 | Hp | 1 |
| ✓ | Assert | alu._assert_11 | Hp (1) | Infinite |
| ✓ | Cover (related) | alu._assert_11:precondition1 | Hp | 1 |
| ✓ | Assert | alu._assert_12 | Hp (1) | Infinite |
| ✓ | Cover (related) | alu._assert_12:precondition1 | Hp | 1 |
| ✓ | Assert | alu._assert_13 | Hp (1) | Infinite |
| ✓ | Cover (related) | alu._assert_13:precondition1 | Hp | 1 |
| ✓ | Assert | alu._assert_14 | Hp (1) | Infinite |
| ✓ | Cover (related) | alu._assert_14:precondition1 | PRE | 1 |
| ✓ | Assert | alu._assert_15 | Hp (1) | Infinite |
| ✓ | Cover (related) | alu._assert_15:precondition1 | Hp | 1 |
| ✓ | Assert | alu._assert_16 | Hp (1) | Infinite |
| ✓ | Cover (related) | alu._assert_16:precondition1 | Hp | 1 |

## Controller.sv

### Features



```
module Controller(

    //Input
    input logic [6:0] Opcode, //7-bit opcode field from the instruction

    //Outputs
    output logic ALUSrc,//0: The second ALU operand comes from the second register file output (Read data 2);
                  //1: The second ALU operand is the sign-extended, lower 16 bits of the instruction.
    output logic MemtoReg, //0: The value fed to the register Write data input comes from the ALU.
                     //1: The value fed to the register Write data input comes from the data memory.
    output logic RegWrite, //The register on the Write register input is written with the value on the Write data input
    output logic MemRead,  //Data memory contents designated by the address input are put on the Read data output
    output logic MemWrite, //Data memory contents designated by the address input are replaced by the value on the Write data input.
    output logic [1:0] ALUOp,   //00: LW/SW/AUIPC; 01:Branch; 10: Rtype/Itype; 11:JAL/LUI
    output logic Branch,  //0: branch is not taken; 1: branch is taken
    output logic JalrSel,      //0: Jalr is not taken; 1: jalr is taken
    output logic [1:0] RWSel    //00: Register Write Back; 01: PC+4 write back(JAL/JALR); 10: imm-gen write back(LUI); 11: pc+imm-gen write back(AUIPC)
);
```

# Controller.sv

## Features

ALUSrc: Controls the source of the second ALU operand.
- 0: From the second register file output.
- 1: From the immediate field of the instruction.

MemtoReg: Controls the source of data to be written back to the register.
- 0: From the ALU result.
- 1: From the data memory.

RegWrite: Controls whether to write to the register file.
- 0: Do not write to the register file.
- 1: Write to the register file.

MemRead: Controls whether to read from the data memory.
- 0: Do not read from the data memory.
- 1: Read from the data memory.

MemWrite: Controls whether to write to the data memory.
- 0: Do not write to the data memory.
- 1: Write to the data memory.

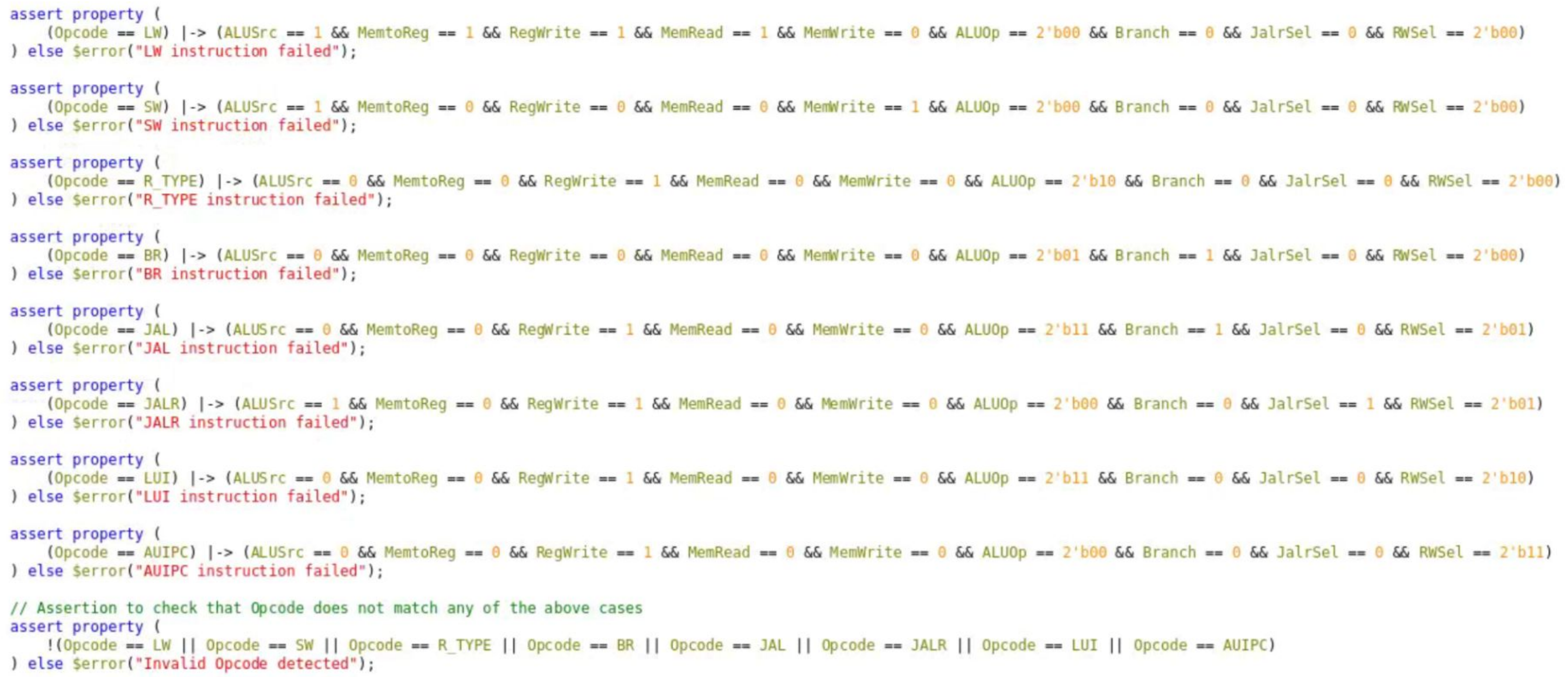
ALUOp: Controls the type of ALU operation.
- 00: LW/SW/AUIPC.
- 01: Branch.
- 10: Rtype/Itype.
- 11: JAL/LUI.

Branch: Controls whether to take a branch.
- 0: Do not take a branch.
- 1: Take a branch.

JalrSel: Controls whether to perform a JALR operation.
- 0: Do not perform a JALR operation.
- 1: Perform a JALR operation.

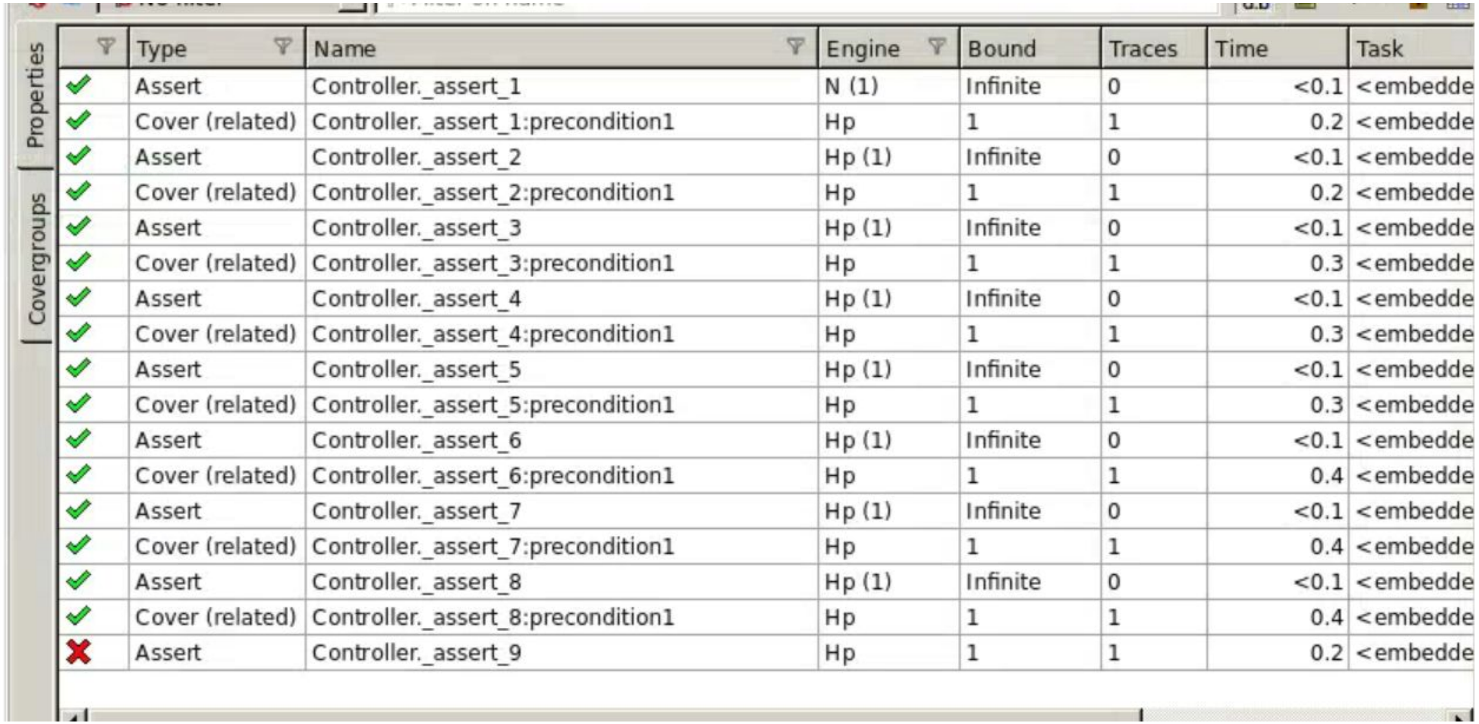
RWSel: Controls the source of data to be written back to the register.
- 00: Register write back.
- 01: PC+4 write back (JAL/JALR).
- 10: Immediate generation write back (LUI).
- 11: PC+immediate generation write back (AUIPC).

# Controller.sv



```
assert property (
    (Opcode == LW) |-> (ALUSrc == 1 && MemtoReg == 1 && RegWrite == 1 && MemRead == 1 && MemWrite == 0 && ALUOp == 2'b00 && Branch == 0 && JalrSel == 0 && RWSel == 2'b00)
) else $error("LW instruction failed");

assert property (
    (Opcode == SW) |-> (ALUSrc == 1 && MemtoReg == 0 && RegWrite == 0 && MemRead == 0 && MemWrite == 1 && ALUOp == 2'b00 && Branch == 0 && JalrSel == 0 && RWSel == 2'b00)
) else $error("SW instruction failed");

assert property (
    (Opcode == R_TYPE) |-> (ALUSrc == 0 && MemtoReg == 0 && RegWrite == 1 && MemRead == 0 && MemWrite == 0 && ALUOp == 2'b10 && Branch == 0 && JalrSel == 0 && RWSel == 2'b00)
) else $error("R_TYPE instruction failed");

assert property (
    (Opcode == BR) |-> (ALUSrc == 0 && MemtoReg == 0 && RegWrite == 0 && MemRead == 0 && MemWrite == 0 && ALUOp == 2'b01 && Branch == 1 && JalrSel == 0 && RWSel == 2'b00)
) else $error("BR instruction failed");

assert property (
    (Opcode == JAL) |-> (ALUSrc == 0 && MemtoReg == 0 && RegWrite == 1 && MemRead == 0 && MemWrite == 0 && ALUOp == 2'b11 && Branch == 1 && JalrSel == 0 && RWSel == 2'b01)
) else $error("JAL instruction failed");

assert property (
    (Opcode == JALR) |-> (ALUSrc == 1 && MemtoReg == 0 && RegWrite == 1 && MemRead == 0 && MemWrite == 0 && ALUOp == 2'b00 && Branch == 0 && JalrSel == 1 && RWSel == 2'b01)
) else $error("JALR instruction failed");

assert property (
    (Opcode == LUI) |-> (ALUSrc == 0 && MemtoReg == 0 && RegWrite == 1 && MemRead == 0 && MemWrite == 0 && ALUOp == 2'b11 && Branch == 0 && JalrSel == 0 && RWSel == 2'b10)
) else $error("LUI instruction failed");

assert property (
    (Opcode == AUIPC) |-> (ALUSrc == 0 && MemtoReg == 0 && RegWrite == 1 && MemRead == 0 && MemWrite == 0 && ALUOp == 2'b00 && Branch == 0 && JalrSel == 0 && RWSel == 2'b11)
) else $error("AUIPC instruction failed");

// Assertion to check that Opcode does not match any of the above cases
assert property (
    !(Opcode == LW || Opcode == SW || Opcode == R_TYPE || Opcode == BR || Opcode == JAL || Opcode == JALR || Opcode == LUI || Opcode == AUIPC)
) else $error("Invalid Opcode detected");
```

# Controller.sv

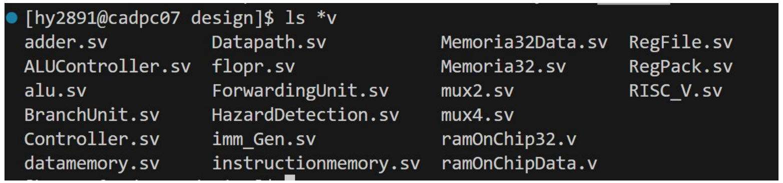


| | Type | Name | Engine | Bound | Traces | Time | Task |
|---|---|---|---|---|---|---|---|
| ✓ | Assert | Controller._assert_1 | N (1) | Infinite | 0 | <0.1 | <embedde |
| ✓ | Cover (related) | Controller._assert_1:precondition1 | Hp | 1 | 1 | 0.2 | <embedde |
| ✓ | Assert | Controller._assert_2 | Hp (1) | Infinite | 0 | <0.1 | <embedde |
| ✓ | Cover (related) | Controller._assert_2:precondition1 | Hp | 1 | 1 | 0.2 | <embedde |
| ✓ | Assert | Controller._assert_3 | Hp (1) | Infinite | 0 | <0.1 | <embedde |
| ✓ | Cover (related) | Controller._assert_3:precondition1 | Hp | 1 | 1 | 0.3 | <embedde |
| ✓ | Assert | Controller._assert_4 | Hp (1) | Infinite | 0 | <0.1 | <embedde |
| ✓ | Cover (related) | Controller._assert_4:precondition1 | Hp | 1 | 1 | 0.3 | <embedde |
| ✓ | Assert | Controller._assert_5 | Hp (1) | Infinite | 0 | <0.1 | <embedde |
| ✓ | Cover (related) | Controller._assert_5:precondition1 | Hp | 1 | 1 | 0.3 | <embedde |
| ✓ | Assert | Controller._assert_6 | Hp (1) | Infinite | 0 | <0.1 | <embedde |
| ✓ | Cover (related) | Controller._assert_6:precondition1 | Hp | 1 | 1 | 0.4 | <embedde |
| ✓ | Assert | Controller._assert_7 | Hp (1) | Infinite | 0 | <0.1 | <embedde |
| ✓ | Cover (related) | Controller._assert_7:precondition1 | Hp | 1 | 1 | 0.4 | <embedde |
| ✓ | Assert | Controller._assert_8 | Hp (1) | Infinite | 0 | <0.1 | <embedde |
| ✓ | Cover (related) | Controller._assert_8:precondition1 | Hp | 1 | 1 | 0.4 | <embedde |
| ✗ | Assert | Controller._assert_9 | Hp | 1 | 1 | 0.2 | <embedde |

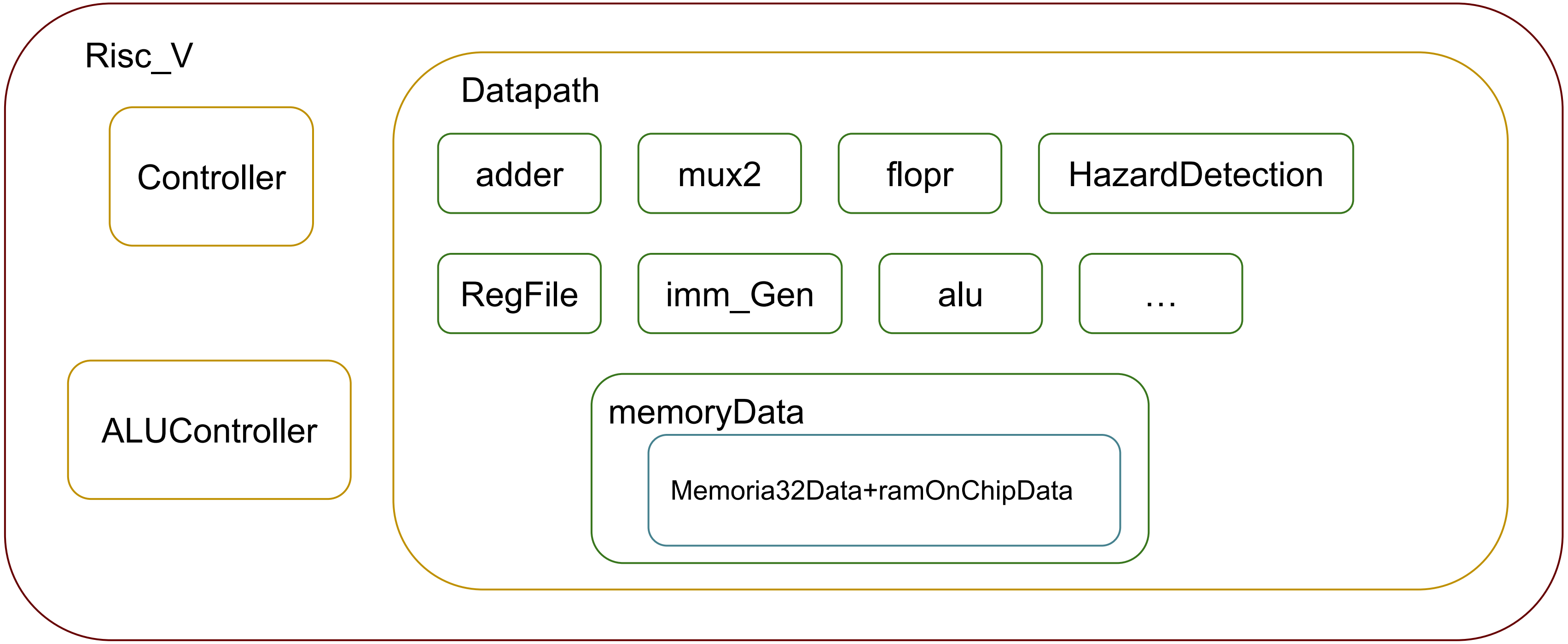
## Why?

Opcode is an input from uplink module. There are implicit limits on it.

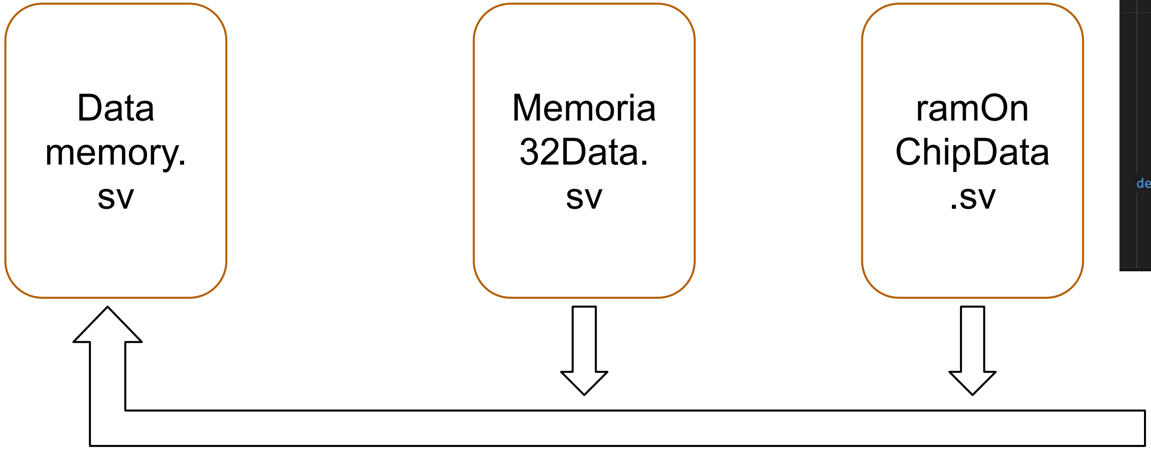
# Attempts To Run Hierarchy Design

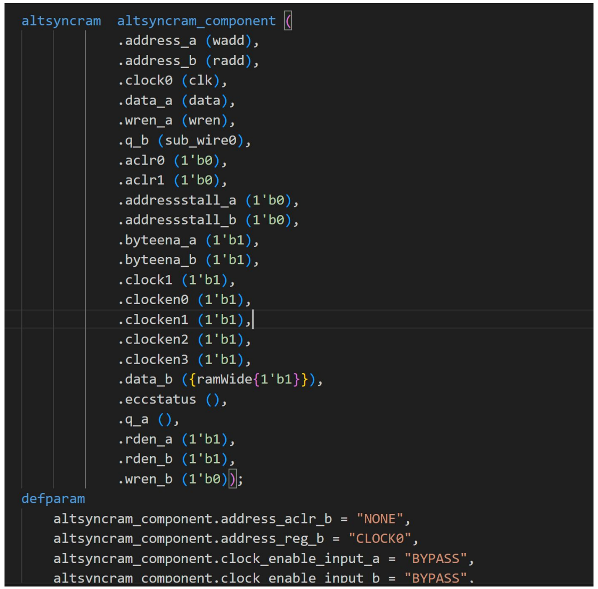


```
[hy2891@cadpc07 design]$ ls *v
adder.sv          Datapath.sv            Memoria32Data.sv  RegFile.sv
ALUController.sv  flopr.sv               Memoria32.sv      RegPack.sv
alu.sv            ForwardingUnit.sv      mux2.sv           RISC_V.sv
BranchUnit.sv     HazardDetection.sv     mux4.sv
Controller.sv     imm_Gen.sv             ramOnChip32.v
datamemory.sv     instructionmemory.sv   ramOnChipData.v
```



# Attempts To Run Hierarchy Design





```
altsyncram  altsyncram_component (
                .address_a (wadd),
                .address_b (radd),
                .clock0 (clk),
                .data_a (data),
                .wren_a (wren),
                .q_b (sub_wire0),
                .aclr0 (1'b0),
                .aclr1 (1'b0),
                .addressstall_a (1'b0),
                .addressstall_b (1'b0),
                .byteena_a (1'b1),
                .byteena_b (1'b1),
                .clock1 (1'b1),
                .clocken0 (1'b1),
                .clocken1 (1'b1),
                .clocken2 (1'b1),
                .clocken3 (1'b1),
                .data_b ({ramWide{1'b1}}),
                .eccstatus (),
                .q_a (),
                .rden_a (1'b1),
                .rden_b (1'b1),
                .wren_b (1'b0));
defparam
    altsyncram_component.address_aclr_b = "NONE",
    altsyncram_component.address_reg_b = "CLOCK0",
    altsyncram_component.clock_enable_input_a = "BYPASS",
    altsyncram_component.clock_enable_input_b = "BYPASS",
```

Primitive-level IP core

**Altera Synchronous RAM**

When using Modelsim, simulation library paths will include Quartus library, which includes the HDL file for altsyncram. Use “-bbox_m altsyncram”

# Attempts To Verify Hierarchy Design

## Feature

MemRead == 1

LB (Load Byte):

```
    Funct3 = 3'b000
    Loads a byte from memory and sign-extends it to 32 bits.
    rd <= {Dataout[7] ? 24'hFFFFFF : 24'b0, Dataout[7:0]};
```

LH (Load Halfword):

```
    Funct3 = 3'b001
    Loads a halfword from memory and sign-extends it to 32 bits.
    rd <= {Dataout[15] ? 16'hFFFF : 16'b0, Dataout[15:0]};
```

LW (Load Word):

```
    Funct3 = 3'b010
    Loads a word from memory.
    rd <= Dataout;
```

LBU (Load Byte Unsigned):

```
    Funct3 = 3'b100
    Loads a byte from memory and zero-extends it to 32 bits.
    rd <= {24'b0, Dataout[7:0]};
```

LHU (Load Halfword Unsigned):

```
    Funct3 = 3'b101
    Loads a halfword from memory and zero-extends it to 32 bits.
    rd <= {16'b0, Dataout[15:0]}
```

Store Operations:

SB (Store Byte):

```
    Funct3 = 3'b000
    Stores a byte to memory.
    Wr <= (a[1:0]==2'b00) ? 4'b0001 : ((a[1:0]==2'b01) ? 4'b0010 :
    ((a[1:0]==2'b10) ? 4'b0100 : 4'b1000));
    Datain <= (a[1:0]==2'b00) ? {{24{1'b0}}, wd[7:0]} :
    ((a[1:0]==2'b01) ? {{16{1'b0}}, {wd[7:0], {8{1'b0}}}} :
    ((a[1:0]==2'b10) ? {{8{1'b0}}, {wd[7:0], {16{1'b0}}}} : {wd[7:0],
    {24{1'b0}}}));
```

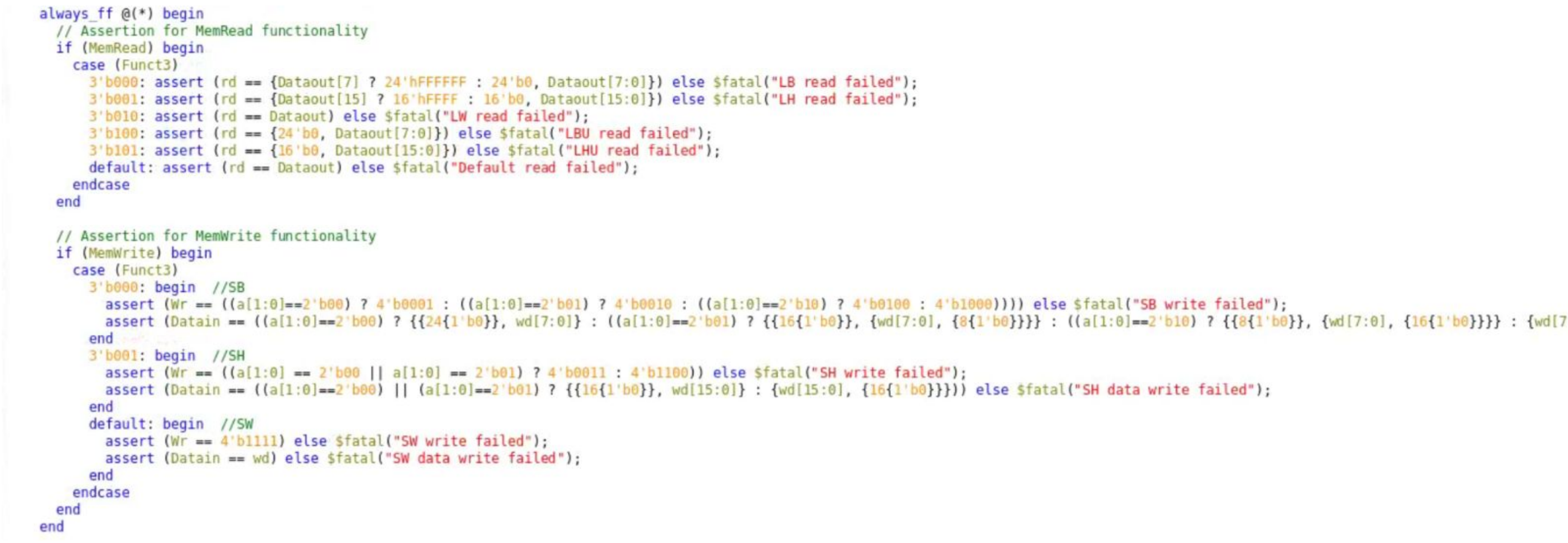
SH (Store Halfword):

```
    Funct3 = 3'b001
    Stores a halfword to memory.
    Wr <= (a[1:0] == 2'b00 || a[1:0] == 2'b01) ? 4'b0011 : 4'b1100;
    Datain <= (a[1:0]==2'b00) || (a[1:0]==2'b01) ? {{16{1'b0}},
    wd[15:0]} : {wd[15:0], {16{1'b0}}};
```

SW (Store Word):

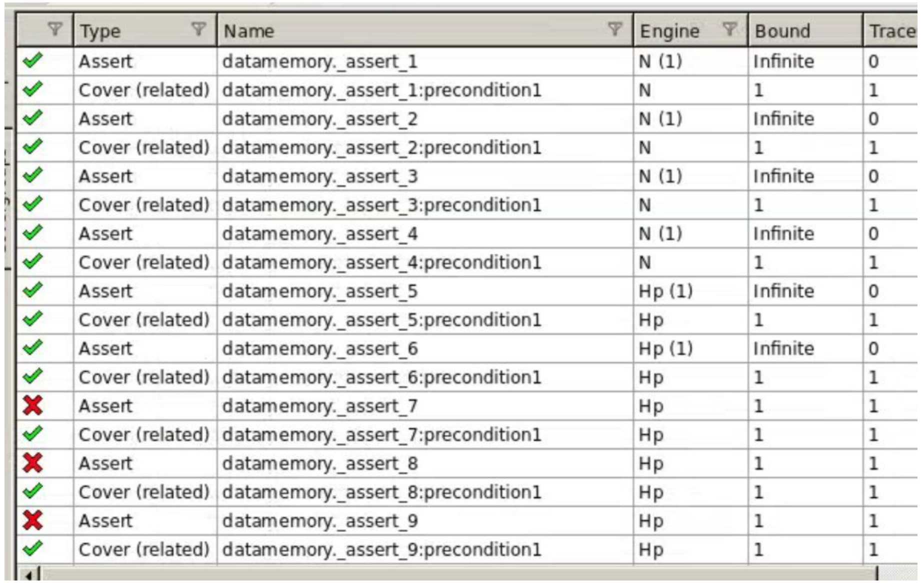
```
    Funct3 = 3'b010
    Stores a word to memory.
    Wr <= 4'b1111;
    Datain <= wd;
```

## Attempts To Verify Hierarchy Design

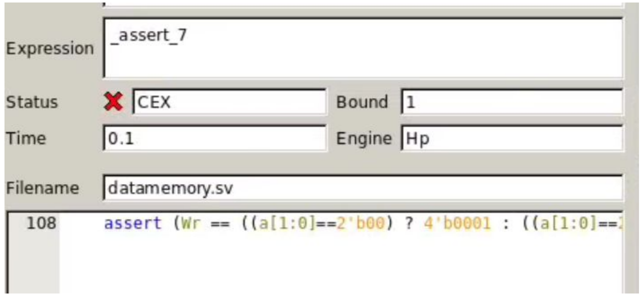


```
always_ff @(*) begin
  // Assertion for MemRead functionality
  if (MemRead) begin
    case (Funct3)
      3'b000: assert (rd == {Dataout[7] ? 24'hFFFFFF : 24'b0, Dataout[7:0]}) else $fatal("LB read failed");
      3'b001: assert (rd == {Dataout[15] ? 16'hFFFF : 16'b0, Dataout[15:0]}) else $fatal("LH read failed");
      3'b010: assert (rd == Dataout) else $fatal("LW read failed");
      3'b100: assert (rd == {24'b0, Dataout[7:0]}) else $fatal("LBU read failed");
      3'b101: assert (rd == {16'b0, Dataout[15:0]}) else $fatal("LHU read failed");
      default: assert (rd == Dataout) else $fatal("Default read failed");
    endcase
  end

  // Assertion for MemWrite functionality
  if (MemWrite) begin
    case (Funct3)
      3'b000: begin  //SB
        assert (Wr == ((a[1:0]==2'b00) ? 4'b0001 : ((a[1:0]==2'b01) ? 4'b0010 : ((a[1:0]==2'b10) ? 4'b0100 : 4'b1000)))) else $fatal("SB write failed");
        assert (Datain == ((a[1:0]==2'b00) ? {{24{1'b0}}, wd[7:0]} : ((a[1:0]==2'b01) ? {{16{1'b0}}, {wd[7:0], {8{1'b0}}}} : ((a[1:0]==2'b10) ? {{8{1'b0}}, {wd[7:0], {16{1'b0}}}} : {wd[7
      end
      3'b001: begin  //SH
        assert (Wr == ((a[1:0] == 2'b00 || a[1:0] == 2'b01) ? 4'b0011 : 4'b1100)) else $fatal("SH write failed");
        assert (Datain == ((a[1:0]==2'b00) || (a[1:0]==2'b01) ? {{16{1'b0}}, wd[15:0]} : {wd[15:0], {16{1'b0}}})) else $fatal("SH data write failed");
      end
      default: begin  //SW
        assert (Wr == 4'b1111) else $fatal("SW write failed");
        assert (Datain == wd) else $fatal("SW data write failed");
      end
    endcase
  end
end
```

# Attempts To Verify Hierarchy Design



| | Type | Name | Engine | Bound | Trace |
|---|---|---|---|---|---|
| ✓ | Assert | datamemory._assert_1 | N (1) | Infinite | 0 |
| ✓ | Cover (related) | datamemory._assert_1:precondition1 | N | 1 | 1 |
| ✓ | Assert | datamemory._assert_2 | N (1) | Infinite | 0 |
| ✓ | Cover (related) | datamemory._assert_2:precondition1 | N | 1 | 1 |
| ✓ | Assert | datamemory._assert_3 | N (1) | Infinite | 0 |
| ✓ | Cover (related) | datamemory._assert_3:precondition1 | N | 1 | 1 |
| ✓ | Assert | datamemory._assert_4 | N (1) | Infinite | 0 |
| ✓ | Cover (related) | datamemory._assert_4:precondition1 | N | 1 | 1 |
| ✓ | Assert | datamemory._assert_5 | Hp (1) | Infinite | 0 |
| ✓ | Cover (related) | datamemory._assert_5:precondition1 | Hp | 1 | 1 |
| ✓ | Assert | datamemory._assert_6 | Hp (1) | Infinite | 0 |
| ✓ | Cover (related) | datamemory._assert_6:precondition1 | Hp | 1 | 1 |
| ✗ | Assert | datamemory._assert_7 | Hp | 1 | 1 |
| ✓ | Cover (related) | datamemory._assert_7:precondition1 | Hp | 1 | 1 |
| ✗ | Assert | datamemory._assert_8 | Hp | 1 | 1 |
| ✓ | Cover (related) | datamemory._assert_8:precondition1 | Hp | 1 | 1 |
| ✗ | Assert | datamemory._assert_9 | Hp | 1 | 1 |
| ✓ | Cover (related) | datamemory._assert_9:precondition1 | Hp | 1 | 1 |



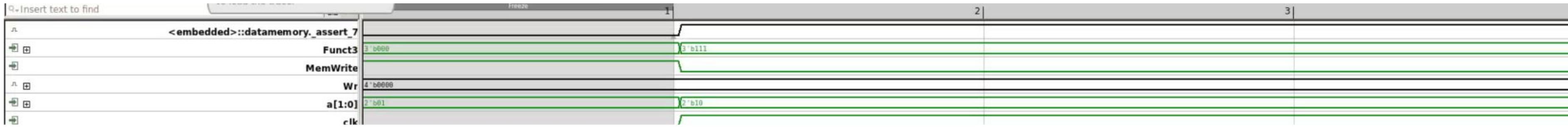
Expression: _assert_7

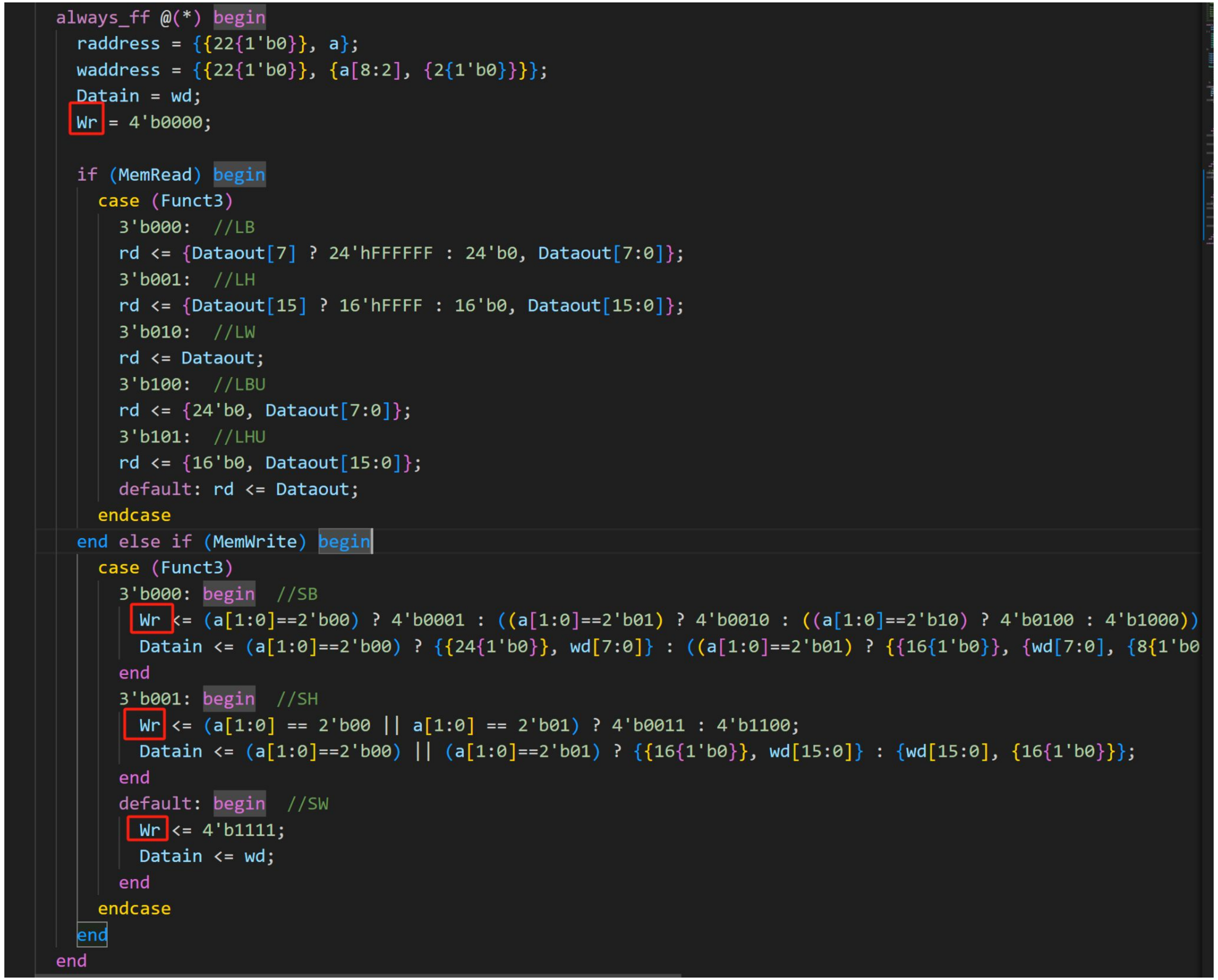
Status: ✗ CEX Bound: 1

Time: 0.1 Engine: Hp

Filename: datamemory.sv

```
assert (Wr == ((a[1:0]==2'b00) ? 4'b0001 : ((a[1:0]==
```

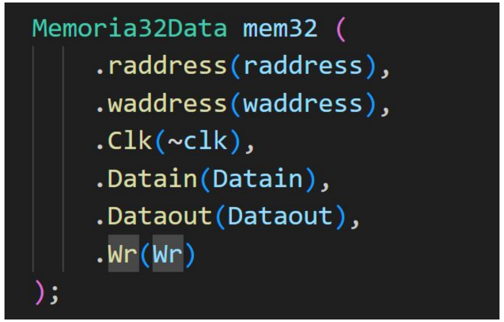




```
always_ff @(*) begin
  raddress = {{22{1'b0}}, a};
  waddress = {{22{1'b0}}, {a[8:2], {2{1'b0}}}};
  Datain = wd;
  Wr = 4'b0000;

  if (MemRead) begin
    case (Funct3)
      3'b000:  //LB
      rd <= {Dataout[7] ? 24'hFFFFFF : 24'b0, Dataout[7:0]};
      3'b001:  //LH
      rd <= {Dataout[15] ? 16'hFFFF : 16'b0, Dataout[15:0]};
      3'b010:  //LW
      rd <= Dataout;
      3'b100:  //LBU
      rd <= {24'b0, Dataout[7:0]};
      3'b101:  //LHU
      rd <= {16'b0, Dataout[15:0]};
      default: rd <= Dataout;
    endcase
  end else if (MemWrite) begin
    case (Funct3)
      3'b000: begin  //SB
        Wr <= (a[1:0]==2'b00) ? 4'b0001 : ((a[1:0]==2'b01) ? 4'b0010 : ((a[1:0]==2'b10) ? 4'b0100 : 4'b1000))
        Datain <= (a[1:0]==2'b00) ? {{24{1'b0}}, wd[7:0]} : ((a[1:0]==2'b01) ? {{16{1'b0}}, {wd[7:0], {8{1'b0
      end
      3'b001: begin  //SH
        Wr <= (a[1:0] == 2'b00 || a[1:0] == 2'b01) ? 4'b0011 : 4'b1100;
        Datain <= (a[1:0]==2'b00) || (a[1:0]==2'b01) ? {{16{1'b0}}, wd[15:0]} : {wd[15:0], {16{1'b0}}};
      end
      default: begin  //SW
        Wr <= 4'b1111;
        Datain <= wd;
      end
    endcase
  end
end
```

Exist non-blocking and blocking assignment in one always block.

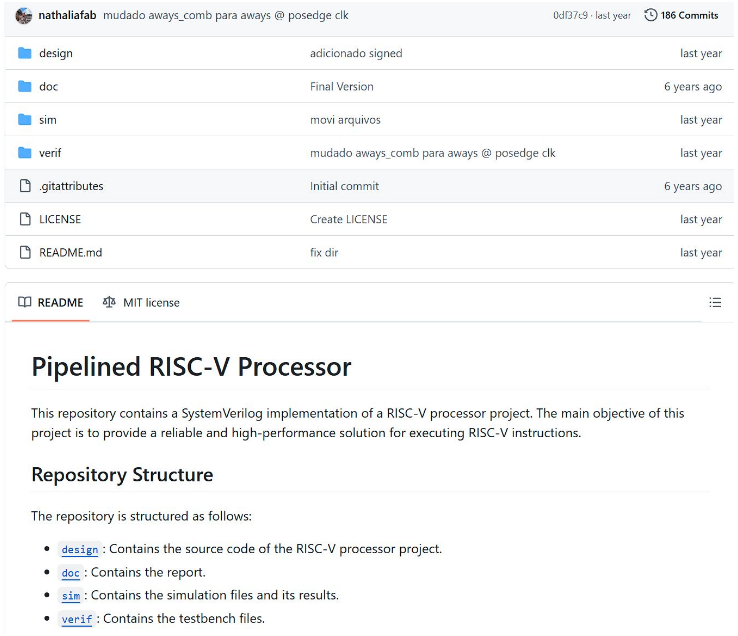
Cannot pass the FV.



```
Memoria32Data mem32 (
    .raddress(raddress),
    .waddress(waddress),
    .Clk(~clk),
    .Datain(Datain),
    .Dataout(Dataout),
    .Wr(Wr)
);
```

# Conclusion

1. Verified the basic block in Risc_V block.
2. Found the design error.



nathaliafab mudado aways_comb para aways @ posedge clk 0df37c9 · last year 186 Commits

| | | |
|---|---|---|
| design | adicionado signed | last year |
| doc | Final Version | 6 years ago |
| sim | movi arquivos | last year |
| verif | mudado aways_comb para aways @ posedge clk | last year |
| .gitattributes | Initial commit | 6 years ago |
| LICENSE | Create LICENSE | last year |
| README.md | fix dir | last year |

## Pipelined RISC-V Processor

This repository contains a SystemVerilog implementation of a RISC-V processor project. The main objective of this project is to provide a reliable and high-performance solution for executing RISC-V instructions.

### Repository Structure

The repository is structured as follows:

- `design` : Contains the source code of the RISC-V processor project.
- `doc` : Contains the report.
- `sim` : Contains the simulation files and its results.
- `verif` : Contains the testbench files.

Formal Verification is far stricter than simulation.

Thank You & Questions